SOFTWARE METRICS PROGRAM FOR RISK ASSESSMENT

Lawrence E. Hyatt
Head, Software Assurance Technology Center
NASA/Goddard Space Flight Center
Greenbelt, MD 20771 USA

Linda H. Rosenberg, Ph.D.
Engineering Section Head
Unisys Government Systems
Greenbelt, MD 20771 USA

Abstract

The Software Assurance Technology Center (SATC) has developed a software metrics program consisting of goals, attributes and metrics to support the assessment of project status, risk, and product quality throughout the life cycle. The objective of the software metrics program is to assess risk areas at each phase of the development life cycle and project them into the future. The software development goals in the metrics program are evaluated by a set of attributes that help to define and classify risks. The attributes must be "measurable" by a set of metrics. These metrics must be based on data that is collectable within the confines of the software development process and must also be relevant to the quality attributes and risk assessment. This paper discusses the SATC's software risk assessment metrics program which meets these needs and is currently being applied to software developed for NASA. At each phase of the software development life cycle, attributes will be identified and metrics defined. Project data is used to demonstrate how the metric analysis was applied at that phase for risk assessment, and how that information could be used by management to manage project risks.

Introduction

The National Aeronautics and Space Agency (NASA) is increasingly reliant on software for the functionality of the systems it develops and uses and recognizes the importance of assessing software risks. This has led to the establishment of the Software Assurance Technology Center (SATC), which, as part of its role of improving the quality of software at Goddard Space Flight Center (GSFC) and other NASA centers, has software metrics as one of its areas of emphasis. Poor quality, especially of products produced early in the life cycle, indicates a high degree of risk when the products are used later in the life cycle. For example, poorly written requirements can result in wrong functionality and cause delays during implementation and testing, putting the schedule at risk.

In order to satisfy the task of evaluating software product quality and potential risks, the SATC has identified a set of attributes that help define and classify the risks. A family of software metrics quantifies the attributes and supports assessment of project status and risk, and product quality throughout the life cycle. The family of metrics concentrates on the following processes or life cycle phases: Requirements, Design, Code and Testing (System and Integration). The SATC metric experience of over 5 years was applied when choosing the attributes and metrics.

Risk, Quality, and Metrics

There is increasing acceptance in the software development community that software metrics are not only useful, but are essential aids in managing a software development project. Metrics can evaluate development processes and the quality of their products.3 Metrics can also assist in risk management and control by identifying areas of possible risk and then to prioritize and track risks.

Risk management is a combination of risk assessment and risk control. Risk assessment includes identification, analysis, and prioritization; risk control includes planning, resolution, and monitoring.1 The project risk management process can be summarized as identification of risks, ranking and prioritization of risks, resolution of those deemed significant, and monitoring risks throughout their applicable life.

In order to identify software metrics that are useful in the risk management process, it is necessary to identify a set of risks that is common to most projects. The risk areas assessed by the SATC are:

- Correctness - Freedom from errors.
- Reliability - Ability to operate over needed time.
- Maintainability - Ease of locating and fixing errors.
- Reusability - Feasibility of reuse.
- Schedule - Developed within specified time.

It should be noted that the risk areas defined above are, in many cases, the same as the attributes of the software defined in the software quality model. Those risks that can be assessed directly are attributes, others, like schedule, are assessed by a combination of attributes and metrics.

Each project manager might have a different prioritization of the listed risks for different segments of software that are being developed by the project. For example, a set of risks relevant to flight software might be, in priority order, too many errors (low reliability), being late (schedule), and not being reusable as a basis for the next flight mission (low reusability). For science analysis software, however, the main risk might be low maintainability (science software changes frequently), with late delivery a secondary risk.

The needs of risk management place two requirements on a metrics program. First, the metrics must support the quantification of the risks - for example, a numeric definition of the number and seriousness of the errors that may be remaining in the software before additional efforts have to be put in place to reduce the risk of low reliability. Second, the metrics must support the overall assessment of the project risk. That is, the risk measures must be capable of being "rolled up" into an overall measure of risk for a given software segment and for the whole project.

SATC Software Quality Model for Risk Assessment

As part of its mission to improve the quality of NASA software, the SATC is assisting software managers in establishing metrics programs that meet their needs with minimal costs, and in interpreting the resulting metrics in the context of the supported projects. Our experience indicates that the project's software risks cannot be evaluated using only software product goals and attributes; goals and attributes for the development process through the life cycle must also be evaluated. Questions of interest are of the nature:

- Can we identify early in the development process the risks to successful completion of the software within schedule?
- Will we achieve adequate correctness/ reliability with the test resources and schedule currently allocated?
- What sections of the code are of the highest reliability and maintainability risk?

In order to convince managers to incorporate a software quality metrics program, we must show them tangible benefits, i.e., increased information about the development process and its risks, increased confidence (reduced risk) that the mission software will be usable when completed, or a real cost saving such as decreased test time. That is, in the ISO terminology, we must select a set of goals that relate to project management.

The SATC has developed a model with goals, associated attributes, and metrics that supports risk management and quality assessment of the processes and products of software development projects. Figure 1 shows the interaction of criteria that are important to NASA software project managers and contained in the SATC model. It contains dynamic elements, so that projections can be made in time to effect the direction of the project.

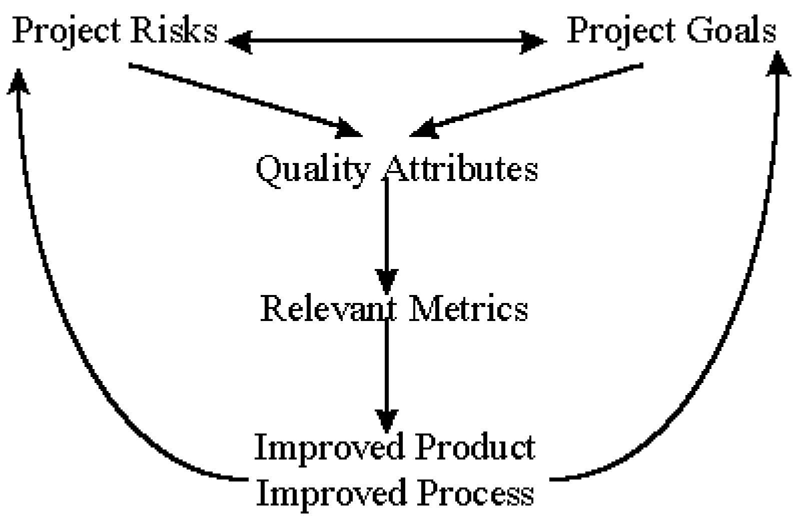


Figure 1: Risk and Quality Interactions

The objective of a software metrics program is to assess the risks at each phase of the development life cycle and to project future risks. Figure 1 diagrams the implementation of this objective. First, risk management - the relevant project risks must be identified, prioritized, then tracked through the life of the project. Second, by identifying and prioritizing project goals, project management will focus the metrics program to specific areas relevant to their project. In order to accomplish this, the metrics program begins with a set of goals that emphasize good processes and quality products and are evaluated by a set of attributes that help to define and classify risks. The attributes are "measurable" by a set of metrics that are computed from data that is possible to collect within the confines of the software development process and that will yield the desired information. In general, there is a one-to-one correlation between the attributes and metrics, but a many-to-many relationship between the attributes and risks. The attributes and metrics do not directly predict the risk severity classification in most cases, determination of the severity for a specific risk may involve a number of metrics, and usually requires an experienced analyst to evaluate the data and assign the risk severity class.

Applying this metrics model leads to two actions: improving the product and improving the process in order to reduce risks. Good processes are necessary but by themselves are insufficient to guarantee a good resulting product; the quality of the product must also be evaluated. Therefore, metric programs should be comprehensive - dynamically evaluate the development process as well as statistically evaluate the products.4

Software Risk Assessment by Life Cycle Phase

In the following sections the attributes and metrics for the phase risks and objectives are defined. It is then demonstrated how these metrics can be used by management to help identify and mitigate risks.

Requirements Phase

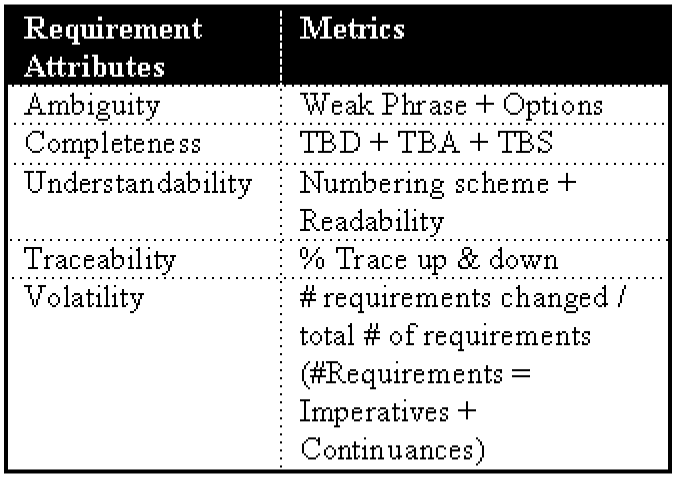
Requirement Attributes and Metrics

The goal for this first phase of software development is a complete, unambiguous, and understandable requirements document, the stabilization of requirements as quickly as possible, and the traceability of all requirements from their source to the software requirements document and then through design and implementation and test. The primary risks are that the wrong software will be developed, that the software will not be completed on schedule, and that the requirements will not be testable. The associated attributes are:

- Ambiguity - Potential multiple meanings.
- Completeness - Items left to be specified.
- Understandability - Readability of the document.
- Volatility - Rate and time within the life cycle changes are made to the requirements.
- Traceability - Traceability of the requirements upward to higher level documents and downward to code and tests.

The SATC is working on methods to measure requirements document quality in much the same way that code quality is measured, that is, by measuring characteristics of the document itself. This measurement philosophy suffers from the problem that a well written document may still have the wrong requirements. However, it does allow an automated and objective evaluation of the document itself. Therefore, the metrics chosen are ones that lend themselves to automated data collection.

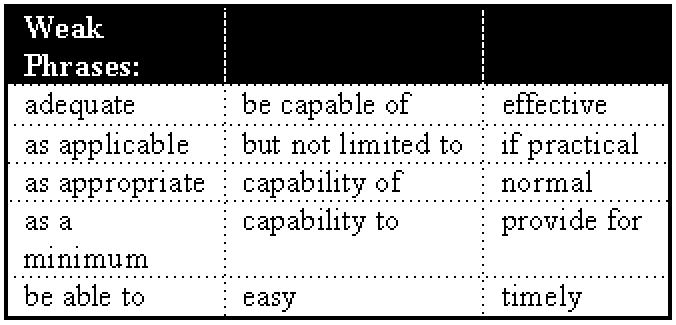
Table 1 Shows the relationships for the requirement attributes and applicable metrics for this phase.



Requirement Attributes	Metrics
Ambiguity	Weak Phrase + Options
Completeness	TBD + TBA + TBS
Understandability	Numbering scheme + Readability
Traceability	% Trace up & down
Volatility	# requirements changed / total # of requirements (#Requirements = Imperatives + Continuances)

Table 1: Requirement Attributes & Metrics

Overall assessment of requirements risk has to blend the impact of all of the requirements metrics. To reduce risk, specific requirements that are ambiguous can be identified and revised. The SATC has identified numerous weak phrases leading to ambiguity as shown in Table 2.7



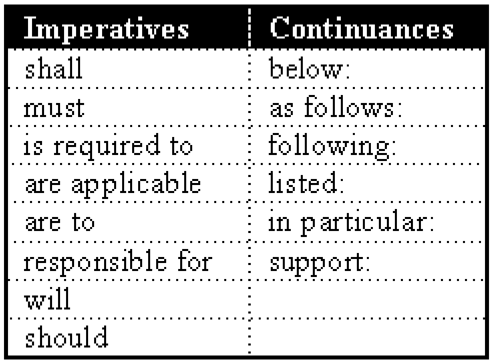
Weak Phrases:		
adequate	be capable of	effective
as applicable	but not limited to	if practical
as appropriate	capability of	normal
as a minimum	capability to	provide for
be able to	easy	timely

Table 2: Weak Phrases

Completeness is measured by the number of items not yet specified- that contain TBAs (to be added), TBDs (to be determined), or TBSs (to be supplied). Measures of Understandability rely on the readability indices, e.g. Flesch - Kincaid, and the number of structures levels within the document and at what level the requirements are specified.7

Volatility is an important factor in risk, and extensive measures are often taken to reduce its impact. But in order to measure volatility, a count of the requirements is needed. Identifying and counting the imperatives and continuances is one method to quantify the number of actual requirements in a document. Table 3 is a list of imperatives and continuances identified by the

SATC as commonly found in requirement documents. The count of the requirements is the sum of the number of imperatives and the number of items that follow continuances.7



Imperatives	Continuances
shall	below:
must	as follows:
is required to	following:
are applicable	listed:
are to	in particular:
responsible for	support:
will	
should	

Table 3: Imperatives & Continuances

Once the requirements have been counted, the volatility can be measured by a count of the number of changes divided by the number of total requirements. The count for the number of changes is not by number of requirements changed, but by applying the same imperative/continuance count used to determine the base number of requirements.

Note - some companies use function points as a measure of the amount of effort needed to complete the tasks. The SATC does not use this metric since no automated tool exists for the actual counting of function points, although many tools exist for the computation of the function point total.

<u>Requirement Risks</u>

It is generally accepted that poorly written, rapidly changing requirements are a principal source of project risk (and indeed, of project failure).The metrics and attributes defined above are related to project risk. For example, the metrics for ambiguity - the count of weak phrases and optional phrases in the requirements document - indicate problems in the requirements document that can result in confusion and the need to take unplanned actions to resolve the questions raised. Thus, the higher the count of ambiguous terms, the higher the project risk.

The requirements counting method discussed above only provides an indicator of the number of requirements. Even though it may not yield an exact count, it does provide the developer a working estimate of tasks to be done. Below is an example of a requirement that would be flagged by the SATC requirements metrics tool as ambiguous (because of the use of "normal speed"). It is also difficult to determine the number of real and implied requirements in the statement, although the tool would only count the one "shall". (In order to preserve the anonymity of the project, ITEM1 or ITEM2 are substituted for project specific terms.)

<u>Requirement</u>: Each channel shall be capable of processing and forwarding telemetry data to the ITEM1 and the ITEM2 at normal speed.

If the imperatives are counted, it has only one task, but in reading the requirement, it becomes obvious that it contains at two or more tasks.

Requirements volatility is always associated with project risk. In addition, the later in the life cycle changes are made to requirements, the more resources needed to implement them. Late requirement changes may also cause a ripple effect, causing additional changes in associated areas. The earlier in the life cycle the requirements stabilize, the lower the risk. It is the SATC experience that the traditional cost increase for the impact of errors shown in Figure 2 can be used for the impact of requirements changes.2

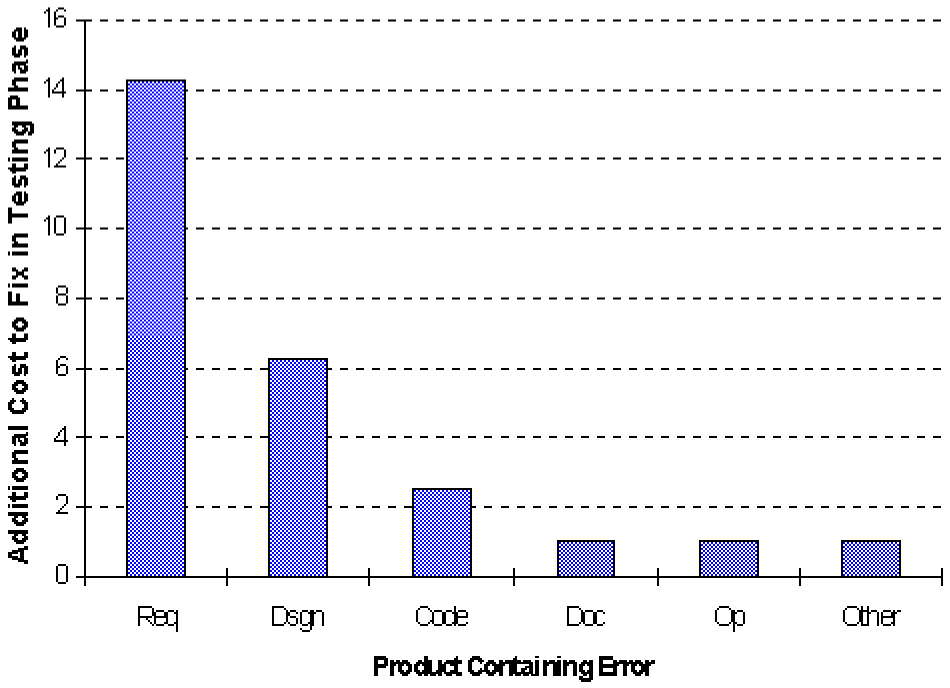


Figure 2: Additional Cost of Errors

Because of the high cost and associated risk of changes made late in the life cycle, requirements attributes and risk must be measured throughout the life cycle.

<u>Design Phase</u>

<u>Design Attributes and Metrics</u>

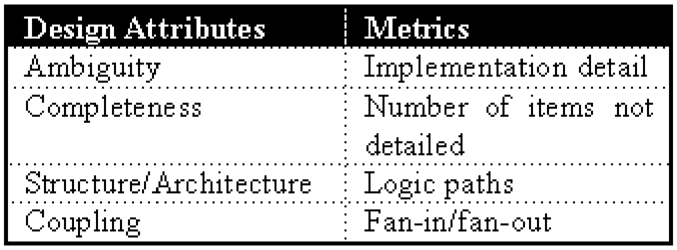
The design process translates requirements into a representation of the software that could be assessed for quality before coding begins. Like requirements, the design is documented and becomes a part of the software configuration baseline. However, the products of the design phase are even more difficult to measure. Design products lack the application of standard depictions and formats and a standard method for developing the design. This makes metrics for the design phase difficult identify and to collect.

Design can be depicted by showing control flow, data flow, or object-oriented/functional structures. An ideal set of metrics covers all three, but most projects are designed by developing only one of these structures. Since most projects develop using control flow structures, the SATC is presenting metrics for this type of development in this paper, although we use metrics for the other types of structures when applicable.

Since the design phase is the transition between requirements and code, the attributes and metrics proposed for design overlap these two phases. The attributes for the design phase are:

- Ambiguity - Multiple meanings.
- Completeness - Items left to be specified.
- Structure/Architecture - Amount of work a module must perform.
- Coupling - Strength of the interconnection from one module to another.

The SATC has identified some design metrics proposed in the industry as shown in Table 4.4 Due to insufficient data, the SATC has been unable at this time to validate this set of metrics. The metrics proposed can, however, be measured in the coding phase and are briefly discussed in that section. Since high complexity increases the propensity for errors and hence increases risks to the schedule and maintainability, the goal of design quality minimizes the complexity of the design. The applicable metrics are level and completeness of the design and two types of complexity, similar to those used for code.



Design Attributes	Metrics
Ambiguity	Implementation detail
Completeness	Number of items not detailed
Structure/Architecture	Logic paths
Coupling	Fan-in/fan-out

Table 4: Design Attributes & Metrics

Ambiguity could be measured by the amount of detail, and Completeness could be measured by the number of modules, not at the lowest level, whose structure is not specified.

Structural complexity is measured using McCabe's cyclomatic complexity. It is a calculation of the number of test paths within a module. Coupling is measured by Fan-in and Fan-out. Fan-in is a count of the calls to a given module. This is the number of local flows into the module plus the number of data structures from which the procedure retrieves information. Fan-out is a count of calls from a given module. This is the number of local flows from a module plus the number of data structures which the module updates.

<u>Design Risks</u>

The design of the software is critical to all future stages of development. If the design is very complex, the coding requires more experienced programmers, more meticulous attention to detail, and more time to code and test. Unfortunately, due to the lack of standard design formats, metrics for this phase are often ignored or omitted from risk evaluation. This increases not only the risk to correctness but can lead to a snowball effect of schedule risk.

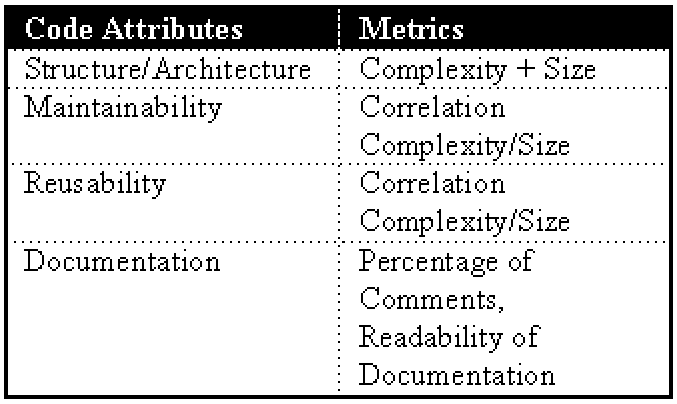
Code Phase

Code Attributes and Metrics

The primary goal of a software development project is to develop code and documentation that will meet the project's requirements. The primary risks in this phase are that the software be maintainable and reusable. The risks to schedule and reliability must be taken into account at this phase, but they are measured in the testing phase. The specific attributes measured during software development are as follows:

- Structure/Architecture - Evaluation of the constructs within a module to identify possible error-prone modules.
- Maintainability - Ease of finding and fixing errors.
- Reusability - Feasibility of reuse of the software
- Documentation - Adequacy and usability of internal and external documentation.

The metrics used to evaluate these attributes are specified in Table 5.



Code Attributes	Metrics
Structure/Architecture	Complexity + Size
Maintainability	Correlation Complexity/Size
Reusability	Correlation Complexity/Size
Documentation	Percentage of Comments, Readability of Documentation

Table 5: Development Attributes & Metrics

The attributes of Structure/Architecture, Maintainability and Reusability use the same metrics for evaluation, but with different emphasis and interpretation guidelines. The three areas of metrics applied here are complexity, size, and the correlation of module complexity with size.

There are many different types of complexity measurements used by the SATC. These include Logical (Cyclomatic) Complexity, which is the number of linearly independent test paths; Data Complexity, which measures the data types and parameter passing; and Calling Complexity, which counts the number of calls to and from modules.5 The extent the GOTO command is used is an indication of disruption of the planned logic flow.

One of the oldest and most common forms of software measurement is size. There are many possibilities for representing size: Lines of Code counts all lines containing program headers, declarations, and executable and non-executable statements; Non-comment Non-blank (Source lines of code) counts lines that are not a comment line or a blank line; and Executable Statements counts the number of executable statements regardless of the number of physical lines.

Code Risks

It is generally accepted that modules with higher complexities are more difficult to understand and have a higher probability of defects than modules with smaller values. Thus complexity has a direct impact on overall quality and specifically on maintainability and reusability. Projects developing code designed for reuse should be especially concerned with complexity since it may later be found to be more expedient to rewrite highly complex modules than to reuse them.

No single metric is a precise risk determinant in the analysis of a product. While poor quality of any product is a risk to the specific goals of the project, some of the best measures of risk come from correlations of the base metrics. Complexity and size of modules are two common measurements of modules, but alone they provide a very one dimensional picture. A more comprehensive view of the data can be obtained by correlating the complexity to the size. A scatter plot of the modules (procedure, function, method, class) is useful to evaluate this data. The SATC uses the Extended Cyclomatic Complexity and the number of Executable Statements. These metrics were chosen because they allow maximum comparison across languages and are the least influenced by programmer style. Using a historical database of code metrics, error data and discussions with numerous programmers and software managers, the SATC developed the diagram applied in Figure 3 (counts in Table 6). Each symbol represents one module. The purpose of this diagram is to help prioritize the modules risk and identify which modules should be further investigated. The seven risk areas are comprehensive, including probability of numerous errors, difficult testing, maintaining and reusing.

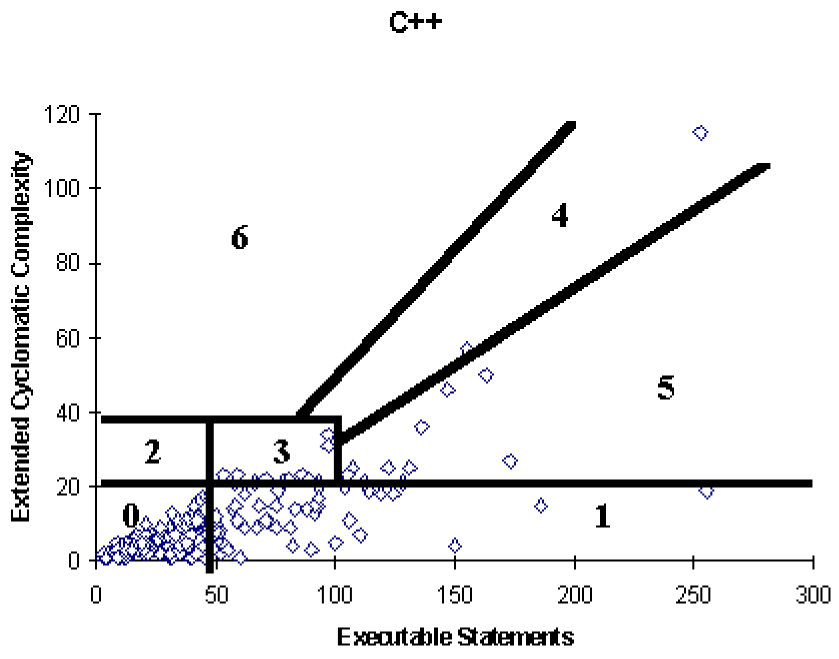
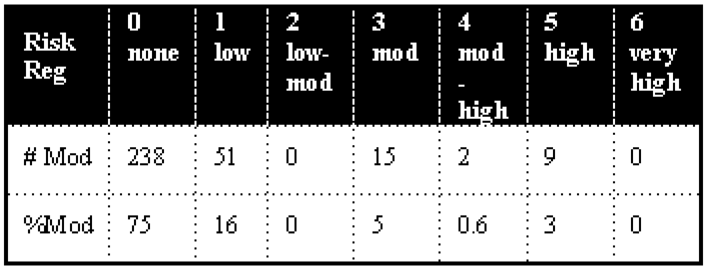


Figure 3: Modules at Risk

The boundaries shown in Figure 3 are guidelines chosen from various industry guidelines and based on SATC experience. Modules with an extended cyclomatic complexity of less than 20 and the number of executable statements less than 50 have a minimal risk (Region 0). These modules are within most industry acceptable ranges. Modules with a complexity less than 20 but more than 50 executable statements are usually lists of assignment statements and are considered low risk (Region 1). Modules with a complexity between 20 and 40 are more difficult to understand. Those with less than 50 executable statements (Region 3) are terse and tend to be more complex and hence a higher risk than those modules with 50 to 100 executable statements (Region 2). The SATC has found that modules with proportional size to complexity, approximately 2 statements for each unit of complexity (Region 4), are easier to re-engineer and tend to be at a lower risk than modules in the two other "HIGH" risk areas. Modules that have a high complexity for a few number of executable statements (Region 6) are the highest risk. These modules are very terse and compact. They tend to be very difficult to understand, increasing the difficulty of maintenance and decreasing the possibility of reuse.

The percentage of modules in each region, shown in Table 6, and a list of the names of the modules in Regions 4, 5, and 6 are supplied to the project management. It is recommended that developers further investigate the modules in these regions using using additional metrics such as fan in/ fan out, comment percentage and number of errors, especially the modules probably at risk in Region 6 with very high complexity and size. One observation made by the SATC is that C and C++ code have a lower percentage of modules in Regions 4, 5 and 6 than does FORTRAN code.



Risk Reg	0 none	1 low	2 low-mod	3 mod	4 mod - high	5 high	6 very high
# Mod	238	51	0	15	2	9	0
%Mod	75	16	0	5	0.6	3	0

Table 6: Counts & Percentages Figure 3

The diagram in Figure 3, when completed, identifies the potential risks to reliability and, especially, maintainability and reusability based on size and complexity. The risk assigned to the areas in the diagram are based on the SATC's experience. The diagram's values are used to show risk for maintainability by the number of modules in the higher risk areas. Lower values for complexity and size would be used to redefine the risk areas for reusability since the candidates for reusability have both upper and lower guidelines.

Since resources are usually limited in any project development, another application of the diagram in Figure 3 is to assist developers and project managers in identifying where to focus resources. Modules in the highest risk areas should be re-engineered if possible, breaking them into smaller units or applying a different design algorithm that decreases complexity. If code redesign and modification is not feasible, high risk modules should be rigorously tested and explicit commenting verified.

Testing Phase

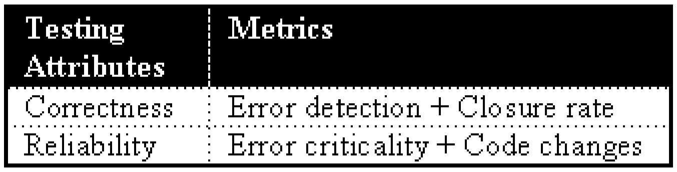
Testing Attributes and Metrics

Once code has been generated and completed unit testing, formal testing - System, Integration, and Acceptance Testing - begins. During formal testing, all software modules are integrated into a cohesive whole and a series of system integration and validation tests are conducted. The purpose of this testing is to find both errors remaining for unit testing and errors which result from unanticipated interactions between subsystems and components. It is also concerned with validating that the overall system provides functions specified in the requirements and that the dynamic characteristics of the system match those required.

The goals for effective testing are to locate and repair faults in the software, to identify error-prone software, and to complete testing on schedule with sufficient number of faults found and repaired that the software will operate as well as needed when it is put into operation. The risks are to schedule and reliability. The attributes measured are:

- Correctness - The likely number of errors remaining in the software.
- Reliability - The likely number of highly critical errors remaining in the software.

The associated metrics are shown in Table 7.



Testing Attributes	Metrics
Correctness	Error detection + Closure rate
Reliability	Error criticality + Code changes

Table 7: Testing Attributes & Metrics

Correctness and Reliability of a computer program are the most important elements of its overall quality. If a program repeatedly and frequently fails to perform, the users will not be satisfied, regardless of the acceptability of other quality factors. Software reliability is defined in statistical terms as the probability of failure free operation of a computer program in a specified environment for a specified time. Software reliability is often measured by the "mean time between failure", but it can also be measured by estimating the number of errors left in the software that are of high criticality - that is, that prevent the software or some major function from operating.6

Correctness can be measured by stating the percentage of errors within a program that must be removed. This implies the ability to estimate the number of errors in the software. One industry guideline is to expect approximately 7 errors per 1000 Source Lines of Code. This estimate is helpful in an overall estimate of the number of errors, but does not take into account the rate at which errors are found. If the rate of finding errors has not started to flatten out, the projection is that if the software is released, the risk to reliability is high that it will not meet the qualifications.

The SATC is working to release the Waterman Error Trending Model for determining the status of testing by projecting of the number of errors remaining in the software and the expected time to find a specified percentage of errors. This model used the Musa Reliability model and the Raleigh curve for effort estimations. The SATC is developing this model rather than using the standard Musa model because it is less sensitive to data inaccuracy and provides for non-constant testing resource levels. Figure 4 is an example of the model's application.

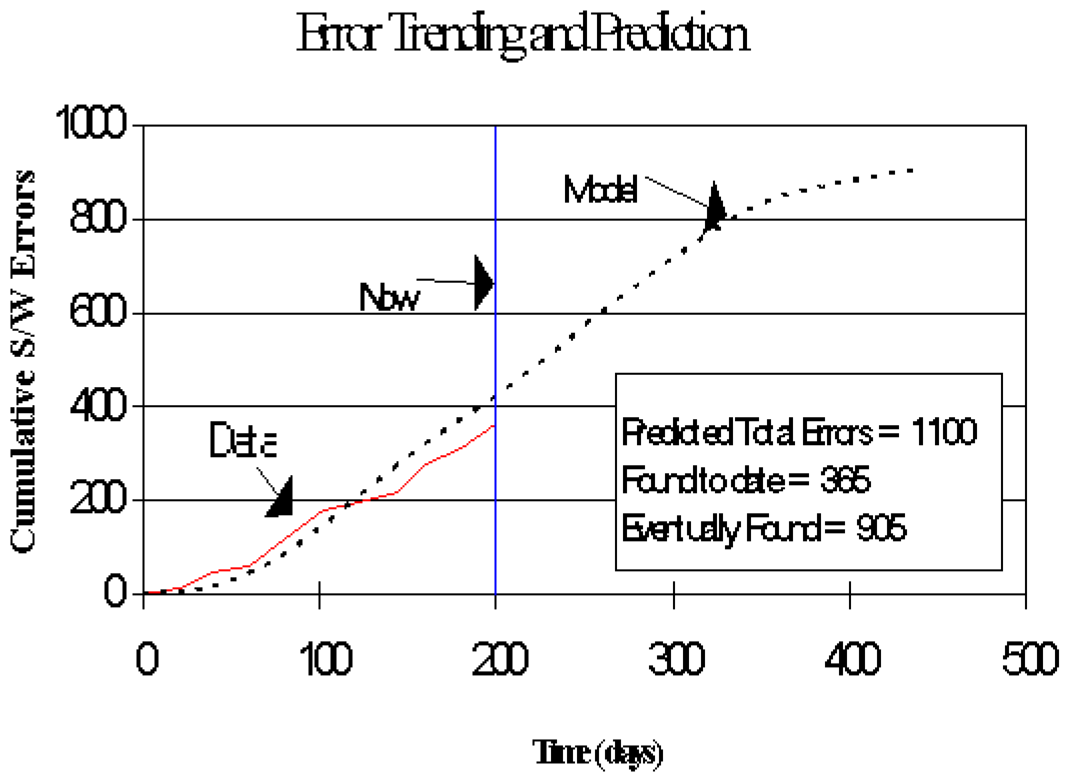
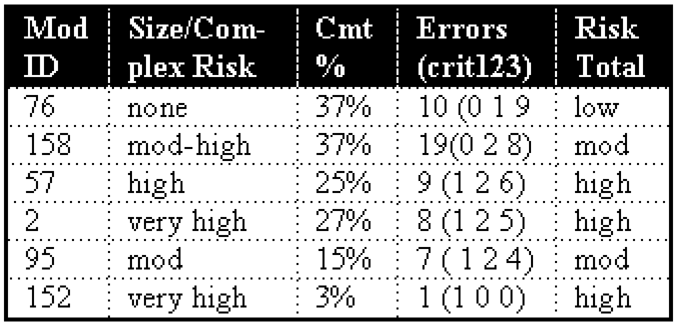


Figure 4: Waterman Error Trending Model

During the testing phase, as the errors are identified, they may result in changes to the code. Code "at risk" that was identified using development phase metrics can now be linked to errors/changes. This linkage is important because changes to code, especially code that is high risk due to size and complexity, can lead to additional problems and ripple effect errors. High risk code with multiple changes before release indicates that re-engineering of that code should strongly be considered. Table 8 indicates a method for combining the metrics on code modules and presenting a risk picture.

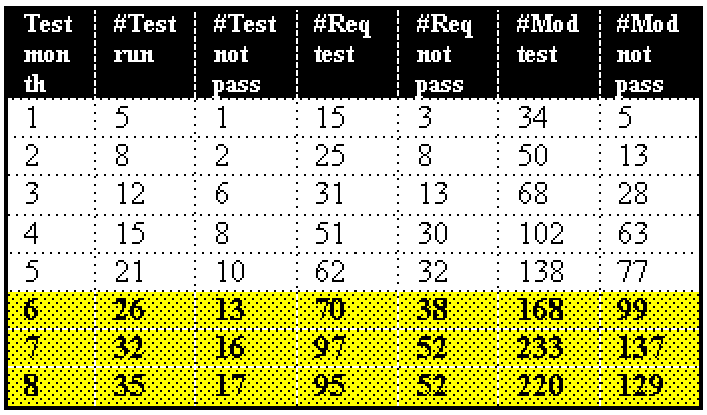


Mod ID	Size/Complex Risk	Cmt %	Errors (crit123)	Risk Total
76	none	37%	10 (0 1 9	low
158	mod-high	37%	19(0 2 8)	mod
57	high	25%	9 (1 2 6)	high
2	very high	27%	8 (1 2 5)	high
95	mod	15%	7 (1 2 4)	mod
152	very high	3%	1 (1 0 0)	high

Table 8: Module Total Risk Indication

In this table, metrics from the development phase are combined with metrics from the testing phase to provide a more accurate assessment of which modules pose potential risks. The size/complexity correlation risk and comment percentage are now combined with the number of errors detected and the criticality of those errors. The comment percentage represents the amount of internal documentation that will provide information to the people making the changes to the code. Minimal internal documentation can cause indirect or "bad" fixes, or introduce new errors. The total risk for these modules is still subjective although the SATC is working to quantify this evaluation process.

Analysis of completion rate data can also be used to identify risks. The completion rate of tests can be used to estimate the risk of completing all tests on time, hence completion of testing requirements and modules. The risk is measured by use of test report data. Table 9 is an example of data that might be compiled from the test data and used for projecting testing results for the next 3 months of testing (test months 6, 7, & 8). Using this information, a project manager might conclude that if all tests must be passed by the end of the eighth month, this project is at high risk.



Test month	#Test run	#Test not pass	#Req test	#Req not pass	#Mod test	#Mod not pass
1	5	1	15	3	34	5
2	8	2	25	8	50	13
3	12	6	31	13	68	28
4	15	8	51	30	102	63
5	21	10	62	32	138	77
6	**26**	**13**	**70**	**38**	**168**	**99**
7	**32**	**16**	**97**	**52**	**233**	**137**
8	**35**	**17**	**95**	**52**	**220**	**129**

Table 9: Sample Test Data and Projections

Testing Risks

The metrics for this phase lend themselves to a number of risk projections. If the project has quantified the percentage of the errors that are to be found before the software is accepted, the Waterman Error Trending Model can be used to project when that will happen. If the time is significantly after the time at which the schedule requires the software to be available, then schedule risk is high.

Another risk determination can be done by looking at the code modules or sub-systems in which the errors occur. Segments of code with more than the average number of errors need investigation to see if re-engineering should be considered. Additionally, the time to fix errors should be tracked. If this metric is increasing, then a threat to reliability and to schedule exists and should be pointed out to the project.

Finally, the location of faults that caused high criticality errors should be tracked. A concentration of them in a segment of code indicates a risk that the requirements are not well understood, or that the design is not suitable.

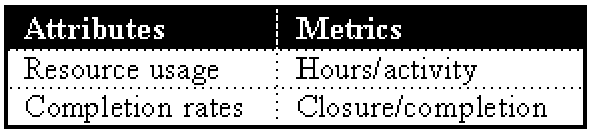
Life Cycle Implementation

Life Cycle Implementation Attributes and Metrics

The goal of implementation effectivity within the life cycle activities is to maximize the effectiveness of resources within the project scheduled activities. This is not phase specific but starts in the requirements phase and continues to the release of the software. The primary risk that can be assessed is to the schedule. The attributes of this goal are:

- Resource use - The extent resource usage correlates to the appropriate phase of the project.
- Completion Rates - Progress made in completing items such as peer reviews or turnover of completed modules to CM.

The metrics for these attributes are shown in Table 10 below.



Attributes	Metrics
Resource usage	Hours/activity
Completion rates	Closure/completion

Table 10: Implementation Attributes & Metrics

The purpose of collecting resource hours is not to monitor personnel attendance but to validate schedule risks. Figure 5 is the Rayleigh curve, which is commonly used to estimate the expected effort expended for any people intensive activity. Projects tend to ramp up fairly quickly then slowly decrease - the level of effort is not consistent throughout implementation and projects need to staff accordingly.2 Projects that schedule a fixed number of personnel throughout the development will be understaffed initially and then over staffed towards the project conclusion.

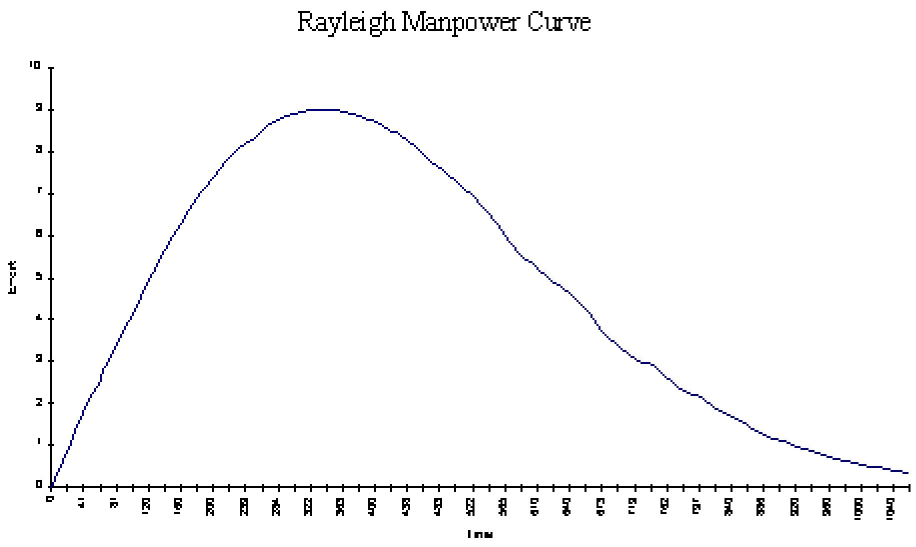


Figure 5: Rayleigh Manpower Effort Curve

Another industry guidelines for expected resource expenditure is 32 hours per week during a 40 hour week. This takes into account time off for vacations, sick leave, and holidays.

The estimate of 32 hours per week and the Rayleigh curve do not take into account the time of year effect however. Although managers expect people to take vacations in the summer and spend time at home with families at Thanksgiving and Christmas, rarely is this decrease in personnel considered when project milestones are determined. Milestones on January 1 have a high risk of not being met if personnel do not work between Christmas and New Years. Even if people do work during that time, their full mental processes may not be on the job. Weather also effects the risk to meeting scheduled milestones. In the Baltimore, Washington area, January and February tend to have ice storms and snow, drastically decreasing available personnel resources.

Life Cycle Implementation Risks

The risks that can be determined from the metrics of resource use are based on the appropriateness of the tasks to which resources are being applied at a given time during the life cycle. To the extent that those activities do not match the expected or planned activities, an element of risk may have been identified. For example, work on prior phase activities during the current phase is a risk indicator. If significant effort is being expended on requirement activities during the design phase (or worse, during the implementation phase), there is significant risk that the project will not be able to meet its schedule objective. This is demonstrated in Figure 6 below.

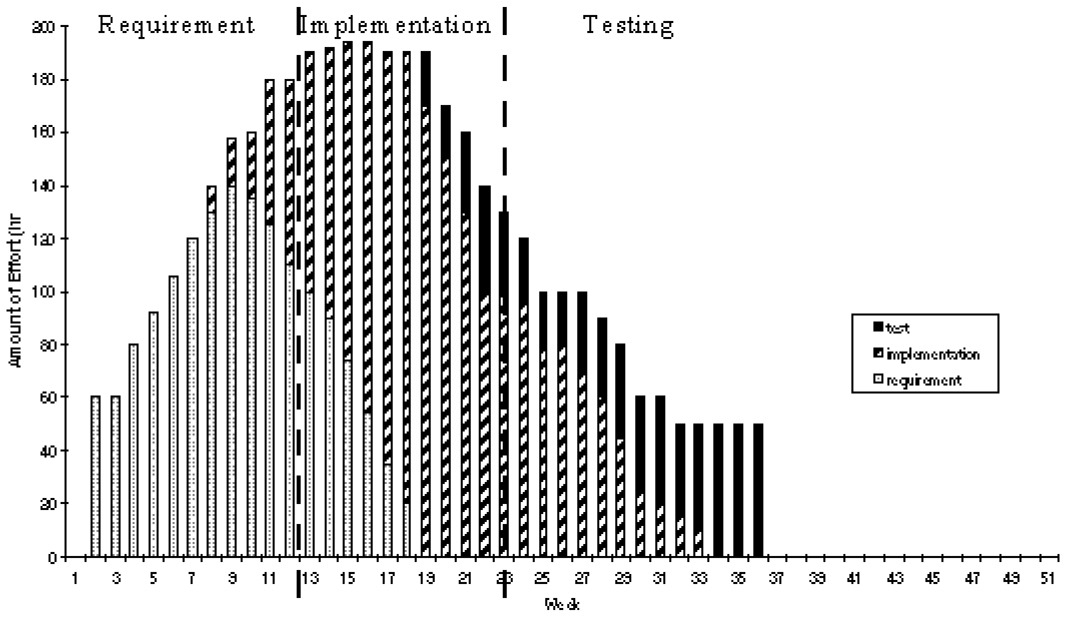


Figure 6: Effort by Activity

In Figure 6 requirements are still being written during the time allotted to coding, rippling coding into the testing phase. This indicates that in order to maintain the schedule, testing must decrease, putting reliability at risk, or the schedule is at risk if the testing must be completed.

Summary

The SATC has applied its metrics experience and some concepts from theoretical models of software quality to develop a unique model for evaluation of quality and project risks. This model fits the needs of project managers in the NASA and GSFC environment in the following ways:

- The model is dynamic, not static, in the fact that it allows the production of multiple snapshots of project status across the development. The data is used to make projects about specific project risks at project milestones.
- The model uses a broad range of measures for both software products and development processes.
- The model is applicable across the development life cycle. The Model's metrics are derived based on aspects of the attributes that answer questions of the project managers. The Model includes analysis guidelines for the data collected.

Future Work

Currently, working with the SATC, software managers and quality assurance engineers are starting to budget for data collection and software metric analysis. Significant effort is being spent on methods to use the metrics to forecast the values of the selected goals and attributes forward to project milestones such as delivery of the software.

Metrics introduced by the SATC, especially those relating to requirements quality need to validated, and a set of higher level design quality metrics is needed to supplement those done at the code level. Our long term objective is to be able to establish a numerical metric scale for assessment of all metrics that affect software quality.

References

1 Boehm, B., *Software Risk Management*, IEEE Computer Society Press, CA, 1989.

2 Fenton, N., *Software Metrics: A Rigorous Approach*, Chapman & Hall, London, UK, 1991.

3 Gillies, A.C., *Software Quality, Theory and Management*, Chapman & Hall, 1992.

4 Hyatt, L. & Rosenberg, L., "A Software Metrics Model for Project Risks and Assessing Quality", European Space Agency Metrics Symposium, 3/96.

5 *IEEE Standard for a Software Quality Metrics Methodology*, IEEE Std 1061-1992, 1992.

6 Kitchenham, B. & Pfleeger, S., "Software Quality: The Elusive Target, *IEEE Software*, 1/96.

7 Wilson, W., Rosenberg, L., Hyatt, L., "Automated Analysis of Requirement Specifications", Fourteenth Annual Pacific Northwest Software Quality Conference, 10/96.

Software Metrics for Risk Assessment was presented at the 47th International Astronautical Congress & Exhibition, 29th Safety and Rescue Symposium, Risk management and Assessment Session, Beijing, China, October 1996.